TAMPEREEN
AMMATTIKORKEAKOULU

JAVASCRIPT VUONNA 2018

Osaaminen ajan tasalle

Lauri Nevanperä

Opinnäytetyö
Toukokuu 2018
Tietojenkäsittely
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Ohjelmistotuotanto

NEVANPERÄ, LAURI:
JavaScript vuonna 2018
Osaaminen ajan tasalle

Opinnäytetyö 43 sivua, joista liitteitä 5 sivua
Toukokuu 2018

Opinnäytetyön tavoitteena oli parantaa sekä yhtenäistää yrityksen JavaScript-osaamista. JavaScript on viime vuosina kehittynyt todella paljon, ja toimeksiantajayrityksessä nähtiin että henkilökunnan kompetenssia tulisi kehittää. Työ toteutettiin kahdessa vaiheessa: Ensimmäisessä vaiheessa opinnäytetyön tekijä opetteli JavaScript-teknologioita ohjelmoimalla tuntikirjausjärjestelmän, ja toisessa vaiheessa järjestelmän teon aikana opittu tieto välitettiin työyhteisölle.

Opinnäytetyön tuloksena syntyi tuntikirjausjärjestelmä. Järjestelmän luomisen jälkeen pidettiin kolmen oppitunnin koulutus nykyaikaisesta JavaScriptistä. Koulutusta tukemaan luotiin sivu toimeksiantajayrityksen sisäiseen tietopankkiin sekä versionhallintarepositorio, jossa on projektipohja, joka sisältää oppitunneilla läpikäydyt kirjastot.

Projekti sujui hyvin. Järjestelmän vapaa ohjelmointi oppimistarkoituksessa osoittautui tekijälle hyväksi ja motivoivaksi tavaksi oppia JavaScript-ekosysteemiä. Annetun koulutuksen hyötyjä ei mitattu, mutta palaute on ollut positiivista ja yrityksen tietopankissa oleva sivusto tarjoaa tulevaisuudessa asiasta tietoa etsiville työntekijöille nopean tavan aloittaa oppiminen.

---

javascript, react, redux, expressjs, mongoose, opettaminen

ABSTRACT

Tampere University of Applied Sciences
Business Information Systems
Software Development

NEVANPERÄ, LAURI:
JavaScript in Year 2018
Improving the Competence

Bachelor's thesis 43 pages, appendices 5 pages
May 2018

The goal of the thesis was to improve the client's JavaScript competency. In the past years, JavaScript has evolved extensively, and therefore the client wanted to improve their employee's JavaScript competence. This was achieved in a 2-phased project. The first phase was to find and learn the new JavaScript technologies by implementing a working time recording service. The second part was to transfer that information to the employees.

As a result of the thesis work, a working time service was created. Three lectures were held and learning resources were created to help further learning. Learning resources included a website containing links to further information and a version control repository containing a boilerplate project with the libraries that were taught during the lectures.

All in all, the project was completed quite well. Implementing the working time recording service turned out to be an effective and motivating way of learning new technologies. It is hard to measure the benefits of the lectures, but the feedback has been positive and in the future the documentation of learning resources will help people to learn.

---

javascript, react, redux, expressjs, mongoose, teaching

SISÄLLYS

LYHENTEET JA TERMIT

JS	JavaScript
ES	EcmaScript
CSS	Cascading Style Sheets

1 JOHDANTO

Tämän opinnäytetyön toimeksiantaja on Sasken Finland. Sasken Finland tekee monenlaisia ohjelmisto- sekä laitteistoprojekteja asiakkaille, sekä tarjoaa konsultointia. Projektit vaihtelevat käyttöliittymäprojekteista laitesuunnitteluun.

Ohjelmistoala on hyvin nopeasti kehittyvä ala, jossa osaamisen tulee olla ajan tasalla. Opinnäytetyön tavoite on kehittää toimeksiantajan työntekijöiden JavaScript-kompetenssia, sekä loiventaa polkuja yrityksen sisäisen osaamisen kehittämiseen. Tarkoituksena opinnäytetyössä on luoda JavaScript-pohjainen tuntikirjausjärjestelmä, ja kehittää tekemisen yhteydessä prosesseja, joilla JavaScript-tietoutta voidaan välittää yrityksessä sisäisesti.

Toimeksiantajalla oli tarve sisäisen tuntikirjausjärjestelmän uudistamiseen. Uudistaminen päätettiin toteuttaa puhtaalta pöydältä moderneja JavaScript-teknologioita käyttäen. Ohjelmoinnin aikana kartutetut tiedot dokumentoitiin ja annettiin työntekijöiden saataville. Järjestelmän uudistamisen jälkeen aloitettiin kuukausittaiset luennot joissa opetettiin järjestelmään valittuja teknologioita. Luentojen tarkoitus on madaltaa työntekijöiden kynnystä aloittaa JavaScript-teknologioiden itsenäinen opiskelu.

JavaScript on ollut ohjelmistomaailmassa viime vuosina paljon pinnalla, ja se lukeutuu myös opinnäytetyön tekijän kiinnostuksen kohteisiin. Trendin sekä tekijän henkilökohtaisen kiinnostuksen vuoksi se valittiin tutkittavaksi ohjelmointikieleksi.

Tässä raportissa käydään läpi edellä mainitun uusitun ohjelmistokokonaisuuden ohjelmoinnin aikana tehtyjä kirjastovalintoja, sekä hyväksi havaittuja käytäntöjä. Itse ohjelmiston toimintaa ei tarkastella syvällisesti. Tässä raportissa kerrotaan myös miten tietoutta välitettiin toimeksiantajan työntekijöille.

Tämän raportin kokonaisvaltainen ymmärtäminen vaatii ohjelmistoalan osaamista.

2 JAVASCRIPT

2.1 Historia

Web-sivut 1990-luvun alussa olivat staattisia, eikä ohjelmoijilla ollut keinoja muokata sivujen sisältöä. Haluttiin animaatioita, interaktiivisuutta sekä muita pieniä asioita jotka parantaisivat web-sivujen tarjoamaa kokemusta. Web-kehitys oli tuohon aikaan yksinkertaista, ja 1990-luvun alun standardeilla tehdyn web-sivun saattoi tehdä vaikka ei tietäisi ohjelmoinnin teoriasta mitään. (Peyrott 2017.)

Java-ohjelmointikieli oli nousussa, ja ajateltiin että tulevaisuudessa suuret, monimutkaiset web-ohjelmat ohjelmoitaisiin Javalla. Tältä ajatuspohjalta vuonna 1995 ruvettiin kehittämään uutta, helppoa ja kevyttä ohjelmointikieltä ei-ohjelmoijlle. Kieltä käytettäisiin monimutkaisen Javan ohessa pienempiin asioihin. Kielen nimeksi tuli JavaScript. (Peyrott 2017.)

Kieli luotiin suuressa kiireessä kilpailevien tekniikoiden pelossa, ja ensimmäinen prototyyppi kielestä saatiin muutamassa viikossa. Tämä prototyyppi lisättiin nopeasti Netscape Communicator –selaimeen. JS sai hyvän vastaanoton ja nykyään lähes jokainen web-sivu sisältää JS:ää. (Peyrott 2017.)

2.2 Historian taakka

Java-ohjelmointikieli webissä ei ottanutkaan tuulta-alleen niin kuin JS:ää luodessa ajateltiin, ja vuonna 2017 raskaan sarjan web-kehitystä tehdään kiireessä suunnitellulla kielellä joka on alunperin suunniteltu helpoksi kieleksi ei-ohjelmoijille. (Peyrott 2017.)

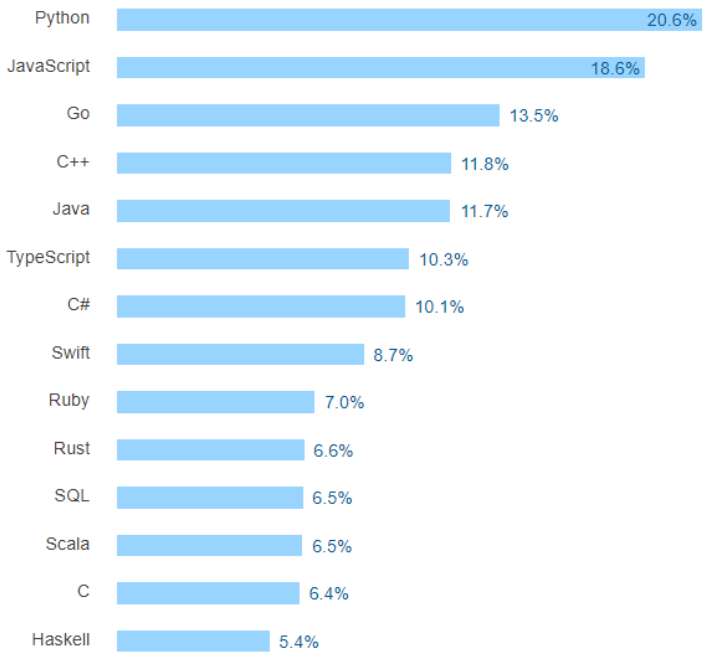
Vaikka JS:ää on kehitetty vuosien varrella paremmaksi, siinä on silti monia epäkohtia (Harvey 2014). Dynaamisesti tyypitettynä kielenä se ei ole hyvin soveltuva suureen projektiin (Jones 2014).

2.3 Vuosi 2018

Miksi sitten selkeistä puutteista huolimatta JS on noussut kehittäjielle tehdyssä suurkyselyssä toiseksi kiinnostavimmaksi ohjelmointikieleksi (Kuvio 1)? Tähän kysymykseen on vaikea löytää yksiselitteistä vastausta.

Kaikki maailman web-selaimet tukevat JS:ää. Tästä johtuu, että puutteistaan huolimatta JS:ää on käytetty todella paljon. Kun valtava joukko kehittäjiä tekee työtä huonolla ohjelmointikielellä, ei ole ihme, että avoimen lähdekoodin JS-kirjastoja on kehittynyt valtavasti helpottamaan kehitystyötä. JS-kirjastoille luodussa pakentinhallintajärjestelmänpm:ssä (Node Packet Manager) on kirjoitushetkellä 350000 JS-kirjastoa (Brown 2017). Tämä on kaksi kertaa enemmän kuin toiseksi suurimmassa paketinhallintajärjestelmässä, Java-kirjastoille tarkoitetussa Mavenissa (Brown 2017). Kielen puutteita on myös kierretty tekemällä kymmenittäin kääntäjiä, jotka muuttavat muita ohjelmointikieliä JS:ksi. (CoffeeScript-versionhallintarepositorio).

JS:n menestys ohjelmointikielenä siis perustuu kehittäjäyhteisön työhön kielen hyväksi. Huonoa ohjelmointikieltä on erinäisillä abstraktioilla parannettu, ja työ on selvästi onnistunut sillä JS on kyselyissä suosittu ja haluttu ohjelmointikieli (Kuvio 1).



KUVIO 1 Halutuimmat ohjelmointikielet (Stack Overflow Developer Survey 2017)

2.4 Ongelmat

Nykyaikainen JS-ohjelmointi vaatii paljon kirjastojen sekä kehitystyökalujen tuntemusta. Oppimiskäyrä on jyrkkä. Ennen käyttöliittymäprojektin aloitusta on pohdittava monia asioita, esimerkiksi:

- Käytetäänkö kääntäjää joka muuttaa toisen ohjelmointikielen JS:ksi?
- Mikä kieli valitaan?
- Mikä ohjelmistoviitekehys tai kirjasto valitaan käyttöliittymän tekemiseen?
- Miten tehty sovellus sekä kirjastot pakataan tehokkaasti muotoon, jossa ne on helppo tarjota selaimelle?

Näihin perusasioihin perehtyinen voi olla vaivalloista nopealiikkeisessä JS-maailmassa, ja siksi kuratoitu tieto on tärkeää.

3 RAKENNETTU JÄRJESTELMÄ YLEISESTI

3.1 Aihe ja tavoite

Projekti alkoi toimeksiantajan tarpeesta uusia työajankirjausjärjestelmä. Opinnäytetyön tekijä oli kiinnostunut JavaScript-tekniikoista, joten järjestelmä päätettiin tehdä täysin JavaScript-tekniikoilla. Järjestelmän teon lisäksi tavoitteena oli hankkia kokemusta uusimmista JavaScript-kirjastoista.

3.2 Suunnittelu

Järjestelmän suunnittelua varten kerättiin tietoa haastattelemalla työyhteisön jäseniä, sekä pitämällä palavereita. Käyttöliittymän yksityiskohtia suunniteltiin kolmenkeskisissä palavereissa, jossa oli opinnäytetyön tekijä, käyttöliittymäsuunnittelija sekä opinnäytetyön tekijän esimies.

Haastattelutavaksi valikoitui avoin ryhmähaastattelu. Tämä haastattelumuoto valittiin, koska jokainen työyhteisön jäsen käyttää tuntikirjausjärjestelmää ja täten omaa hyvän tuntemuksen aiheesta. Haastatteluista kirjattiin ehdotuksia ylös, ja ehdotuksia käytiin läpi opinnäytetyön tekijän esimiehen kanssa. Ehdotuksia arvioitaessa otettiin huomioon ehdotuksen toteuttamiseen menevä työaika sekä siitä saatava hyöty järjestelmään.

3.3 Ominaisuudet

Käyttöliittymä on yhteydettömyys ensin -lähtökohdasta suunniteltu web-sovellus. Sovellukseen sisäänkirjaudutaan toimeksiantajan domain-tunnuksilla. Käyttöliittymästä on mahdollista tallentaa reaaliaikaisesti työaikaa, mikäli käyttäjä niin haluaa. Vaihtoehtoisesti käyttäjä voi lisätä tuntimerkintänsä työsuoritusten jälkeen. Työaikamerkintöjä voi muokata, poistaa, yhdistää ja lisätä. Käyttöliittymässä olevasta graafista voi seurata päivä- ja viikkotyöajan kertymistä.

Käyttöliittymä kommunikoi palvelimen kanssa REST-rajapinnan välityksellä. Palvelinohjelmisto tallentaa käyttäjän tekemät projektikohtaiset tuntimerkinnät sekä pitää listaa esimiehen käyttäjälle merkitsemistä projekteista. Palvelimeen on toteutettu käyttäjän kirjautuminen tunnuksilla, jotka ovat jaettu toimeksiantajan muitten järjestelmien kanssa. Palvelin välittää tallennetut työaikatiedot palkanlaskentajärjestelmälle käyttäjän niin halutessa.

4 KÄYTTÖLIITTYMÄN TOIMINNALLISUUS

Oletusnäkymä

Käyttöliittymän oletusnäkymässä on graafi, joka näyttää päivän sekä viikon kertyneet työtunnit sekä tavoitetyötunnit. Täältä käyttäjä voi asettaa sovelluksen tallentamaan työaikaa tietylle projektille. Tallentaminen tapahtuu play/stop-tyyppisillä kontrolleilla. Esimerkiksi aamulla töihin tullessa painetaan play, jolloin sovellus kysyy, mille projektille tulevat tunnit tallennetaan. Ruokatauolla tallennus pysäytetään, ja tauolta palatessa tallennusta jatketaan. Tunnit tallentuvat verkkoon käyttäjän tilille, joten yhdellä laitteella tallennetut tunnit ovat käytössä kaikilla laitteilla. Kukaan muu ei vielä tässä vaiheessa näe käyttäjän tallennettuja tunteja, vaan ne on tallennettu palvelimella tilaan, jonka tietoihin muilla ei ole pääsyä. (Liite 1)

Listanäkymä

Listanäkymässä käyttäjä voi lisätä, muokata, yhdistää sekä jakaa tuntimerkintöjä. Eri projekteille tehdyt tunnit ovat merkitty eri väreillä. Tästä näkymästä kerrytetyt työtunnit myös lähetetään eteenpäin. Järjestelmä näyttää jo hyväksyttäväksi lähetetyt tunnit harmaina. Näitä tunteja ei voi enää muokata. (Liite 2, Liite 3, Liite 4)

Asetusnäkymä

Asetusnäkymässä käyttäjä voi asettaa viikoittaisen tuntitavoitteen ja -työpäivämäärän, jotka vaikuttavat oletussivun graafin toimintaan. Tällä sivulla voi myös asettaa sovelluksen teeman tummaksi tai vaaleaksi. (Liite 5)

5 KIRJASTOJEN VALITSEMINEN

Kirjastojen valitseminen on projekteissa tärkeä vaihe. Kirjastovalinnat vaikuttavat seuraaviin asioihin:

- Mikä on helppoa toteuttaa ja mikä vaikeaa
- Millä arkkitehtuurilla ohjelmaa kehitetään
- Miten käyttöön löytyy tukea internetistä
- Kuinka nopeasti ohjelma suoriutuu eri tehtävistä

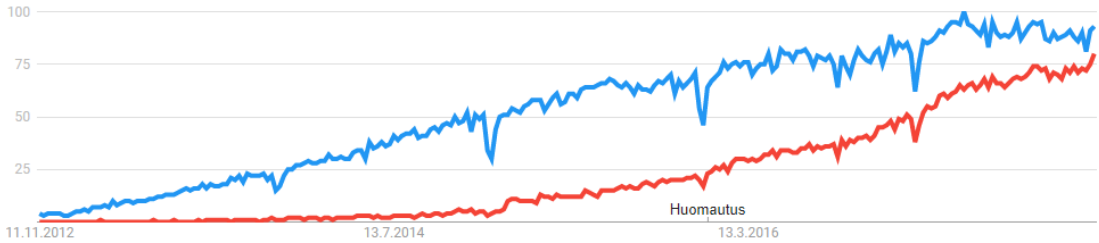
5.1 Valitsemisprosessi

Monet kirjastovalinnat tehtiin kokemuspohjaisesti, mutta mikäli oli tarve tarkkaan arviointiin, tehtiin valinta yhdistelemällä kvantitatiiviisia ja kvalitatiivisia menetelmiä.

Kirjastovalintaprosessi aloitettiin kvalitatiivisesti. Etsittiin Google-haulla hakusanalla ’aihepiiri + JS library’. Tämä haku tuotti linkkejä suosituslistoille, sekä yksittäisten kirjastojen verkkosivuille. Tämän tiedon pohjalta tehtiin lista kirjastoehdokkaista. Listan kirjastoista tarkistettiin soveltuvuus projektiin, sekä API. Soveltuvimpien kirjastoehdokkaitten github-sivuilla (versionhallintapalvelu) käytiin. Github-sivulla tarkistettiin projektille annetut tähdet, avoinna olevat ongelmat sekä päivitysaktiivisuus. Github-tietojen pohjalta karsittiin parhailta vaikuttavat kirjastot seuraavaan, kvalitatiiviseen vaiheeseen.

Kun kvalitatiivisesti on löydetty parhaat vaihtoehdot, aloitettiin kvantitatiivinen osuus. Kirjastojen nimillä etsittiin hakustatistiikat trends.google.com –palvelusta. Tulosten tulkinnassa keskityttiin kokonaishakumäärien sijaan trendiin.

Kuvio 2:n esimerkissä verrataan Angular-käyttöliittymäviitekehyksen (sininen) sekä React- käyttöliittymäkirjaston (punainen) hakumääriä viiden vuoden ajalta. Angular-kirjasto julkaistiin vuonna 2010 (Angular-versionhallintarepositorio), kun taas React-kirjasto julkaistiin vuonna 2012 (React-versionhallintarepositorio). Angular-kirjastolla oli siis kahden vuoden etumatka, ja kuten graafista (Kuvio 2) näkee, sen hakumäärät ovat historiallisesti suurempia. Käyristä näkee että Angular-kirjaston kasvu on tasaista ja se on tasaantunut viimeisen vuoden aikana. React-kirjaston haut ovat taas nousemassa tällä hetkellä voimakkaammin. Tästä johtuen käyttöliittymäkirjastoksi valittiin React.



KUVIO 2. Angular - React -hakusanojen Googlehakujen määrän vertailu ohjelmointi-kategoriassa (trends.google.com)

6 KÄYTTÖLIITTYMÄN JS-KIRJASTOVALINNAT

6.1 React

React on Facebookin kehittämä, avoimen lähdekoodin käyttöliittymäkirjasto. React-ohjelmointi on deklaratiivistä, eli ohjelma muuttaa suoraviivaisesti tilan näkymäksi.

Hieman yksinkertaistaen, Reactissa on kolme tärkeää konseptia (Abramov 2015):

- Elementti: JSON-objekti, joka edustaa html-elementtiä esimerkiksi div tai h1 ja elementin lapsi-sekä lapsenlapsielementtejä sekä attribuutteja.
- Komponentti: Funktio, joka ottaa vastaan parametreja ja koostaa näistä React-elementin.
- Virtuaalinen DOM: Ottaa vastaan React-elementtejä ja muuttaa selaimen näkymää, mikäli vastaanotetut elementit ovat muuttuneet suhteessa edellisiin vastaanotettuihin elementteihin

Komponentille voi antaa parametreiksi lapsielementtejä, jolloin syntyy hyvin html-rakenteen omainen funktiopuu, joka evaluoituu lopulta yhdeksi React-elementiksi, joka on koostettu ohjelman kulloisenkin tilan perusteella. Tämä elementti muutetaan selaimen näkymäksi Virtual DOM:in avulla

React-komponentit ovat siis funktioita, jotka palauttavat html-esityksen itsestään. Tämä tarkoittaa, että ohjelmakoodin rakenne on hyvin html-tyylistä.

Koodiesimerkissä 1 luodaan React-komponentti, joka evaluoituu React-elementiksi, joka on div-tyyppiä, taustaväriltään annetun värin värinen ja sisältää lapsinaan annetut elementit. React-komponentteja käytetään siis hyvin html-tyylisesti. Yllä oleva syntaksi kuitenkin on varsin sekavan näköistä.

```
const ColoredBgComponent = (props) =>
  React.createElement(“div”,
    {style: {backgroundColor: props.color}},
    props.children
  );

ColoredBgComponent({
  {color: ‘red’},
  [ React.createElement(“h1”, {}, ”Title”),
    React.createElement(“p”, {}, “Paragraph”)
  ];
```

KOODIESIMERKKI 1: React-komponentin luonti ja käyttö

Syntaksista sekavuutta helpottamaan on luotu JSX-abstraktio, joka muuttaa React-komponenttien funktiokutsut erilaiseen, HTML-tyyliseen syntaksiin. Edeltävä koodiesimerkki voidaan JSX:n avulla kirjoittaa seuraavasti (Koodiesimerkki 2):

```
const ColoredBgComponent = (props) =>
  <div style={backgroundColor: props.color}>
    {props.children}
  </div>

<ColoredBgComponent color=”red”>
  <h1> Title </h1>
  <p> Paragraph </P>
</ColoredBgComponent>
```

KOODIESIMERKKI 2: React-komponentin luonti ja käyttö JSX-abstraktiolla

React mahdollistaa ohjelman tilan säilyttämisen komponenttien sisällä, mutta tässä projektissa käytettiin erillistä tilanhallintakirjastoa.

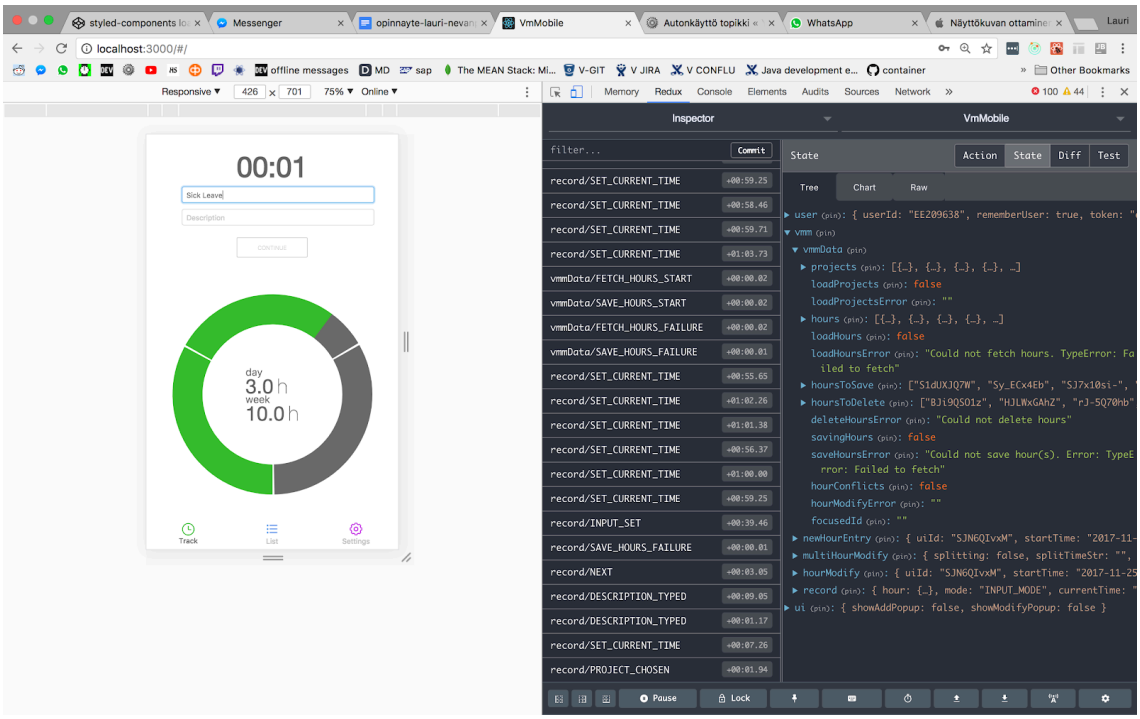
6.2 Redux

Redux on funktionaalinen tilanhallintakirjasto, joka muistuttaa suurelta osin Elm-ohjelmointikielen tilanhallinta-arkkitehtuuria (Redux-repositorio 2018).

Redux voidaan jakaa karkeasti kolmeen pääkonseptiin (Managing your React state with Redux):

- Kauppa (store): Säilyttää tilan
- Tapahtuma (action): Objekti, joka sisältää tapahtumatyypin sekä mahdollisen lisätiedon, jota käytetään tilan modifioimiseen
- Vähennin (reducer): Funktio, joka ottaa vastaan kulloisenkin tilaobjektin sekä tapahtuma-objektin, ja palauttaa uuden tilaobjektin, joka on derivoitu vanhan tilaobjektin sekä tapahtuman pohjalta

Reduxin arkkitehtuuri sekä funktionaalinen luonne tekee siitä hyvin helpon testattavan. Jokainen vaihtunut tila-objekti voidaan liittää kuhunkin lähetettyyn tapahtumaan, ja täten koko sovelluksen tilahistoria voidaan säilyttää. Virhetilanteissa tilahistoria voidaan tallentaa ja sitä voidaan tutkia myöhemmin. Redux DevTools (Kuva 1) on selainlisäosa, jossa ohjelman tilahistoriaa, tapahtumia ja tapahtumien välillä muuttuvia tiloja voi tarkastella reaaliajassa selaimessa. Tästä lisäosasta oli suuri hyöty ohjelmiston virheiden etsimisessä. (Medium 2016a.)



KUVA 1. Redux DevTools -lisäosa

6.3 Redux-persist

Redux-persist on Redux-lisäosa, joka automaattisesti tallentaa ohjelman tilan ohjelmoijan määrittelemään tietokantaan, esimerkiksi JS:n Window.localStorage-kantaan. Tämä tila säilyy kannassa, kun ohjelma sulkeutuu. Kun ohjelmaa käynnistetään seuraavan kerran, Redux-persist hakee tietokannasta ohjelman tilaobjektin ja laukaisee tapahtuman, jonka käyttäjä voi vähentimissä käsitellä haluamallaan tavalla, ja näin hyväksikäyttää edellisen istunnon dataa. (Newton 2017.)

Sovellusta tehtäessä Redux-persist oli olennainen osa yhteydettömyys ensin -kokemuksen aikaansaamisessa. Sen avulla käyttäjän istunto voidaan aina alustaa siihen tilaan, johon edellinen istunto jäi. Esimerkkinä offline-tilassa tallennettu tuntikirjaus, joka säilyy laitteessa eri istuntojen välillä, vaikka sitä ei ole voitu lähettää palvelimelle.

6.4 Reselect

Redux-tilassa tulee säilyttää mahdollisimman vähän dataa, ja kaikki mikä voidaan laskea datan perusteella, tulee laskea (Aier 2016). Valitsin (selector) on funktio, joka ottaa parametriksi ohjelman tilan ja palauttaa tilan perusteella halutun arvon.

Koodiesimerkki 3:ssa esitetään yksinkertainen valitsin. Esimerkissä kuitenkin on yksi ongelma: aina kun valitsimen tulos halutaan, täytyy valitsinoperaatio tehdä uudelleen. Reselect on luotu ratkaisemaan tämä suorituskykyyn liittyvä ongelma. Reselect tallentaa lasketun arvon välimuistiin ja muuttaa sitä vain siinä tapauksessa, että valitsimen tulokseen liittyvä tila on muuttunut. (Reselect-versionhallintarepositiorio.)

```
const isEverythingOkSelector = (state) =>
  state.something.ok && state.somethingElse.ok;
```

KOODIESIMERKKI 3: Valitsin-esimerkki

6.5 Styled-components

Styled-components on React-ohjelmistokirjastolle luotu tyylityskirjasto. Se mahdollistaa Cascading Style Sheets (CSS) -tyylien dynaamisen muuntamisen.

Alkuperäinen tapa tyylitellä HTML-elementtejä on luoda erilliseen CSS-tiedostoon luokkia, joihin on koottu CSS-attribuutteja. Html-elementit tyylitellään asettamalla näille class-attribuutiksi halutun CSS-luokan nimi tekstimuodossa. React-komponenttien tyylien asettaminen toimii samalla tavalla, mutta class-attribuutin sijaan komponentit tyylitellään asettamalla className-propertyksi CSS-luokan nimi.

CSS-luokkien luonti ja niiden viittaus tekstimuotoisella nimellä on perinteinen ja laajalti käytetty tekniikka, mutta opinnäytetyön kirjoittajan mielestä se on epäkäytännöllistä ja ennenkaikkea syntaksillisesti epäselkeää Reactin kaltaisessa deklaratiivisessa ympäristössä.

Koodiesimerkissä 4 on perinteinen tapa tehdä dynaamista CSS:ää react-ympäristössä. Esimerkissä on kaksi tiedostoa. Toinen tiedosto sisältää tyylit sekä luokat, toinen sisältää react-komponentin, jossa props-objektissa olevan boolen-tyyppisen red-attribuutin mukaan lisätään tai poistetaan tyylejä palautettavaan elementtiin.

Edellä mainitussa tavassa on muutamia ongelmia:

- Tyylit ovat erillisessä tiedostossa, ja niihin viitataan merkkijono-viitteellä, joka voi olla virheellinen tai päällekkäinen toisen CSS-luokkanimen kanssa.
- Tyypitetyissä JS-varianteissa ei pysty hyppäämään tyyli-deklaraatioon, koska se ei ole JS-muuttuja, vaan merkkijonoviittaus CSS-luokkaan, joka voisi periaatteessa olla missä tahansa tiedostossa.
- React-komponentin syntaksi on sekavahkoa siitä huolimatta että esimerkissä on käytetty syntaksia yksinkertaistavaa ES6 sapluunamerkkijonoa.

```
---- styles.css ----
.red {
  background-color: red;
}
.collapsed {
  height: 0px;
}
.baseClass {
  Width: 100px;
}

---- component.js ----
import ‘./styles.css’;’

const MaybeRedMaybeCollapsedComponent =
  (props) => {
    const red = props.red ? ‘red’ : ‘’;
    const collapsed = props.collapsed
      ? ‘collapsed’
      : ‘’;

    return
      <span
        className={`baseClass ${red
          } ${collapsed}`
      />;
};
```

KOODIESIMERKKI 4: Dynaaminen CSS Reactissa, perinteinen tapa

Edellä mainittuja ongelmia voi poistaa tai helpottaa apukirjastoilla ja käyttämällä CSS-esiprosessoria. Nämä kuitenkin monimutkaistavat projektia ja kasvattavat oppimiskynnystä. Ne eivät pureudu ongelman alkuperäiseen syyhyn, eli siihen että CSS- ja JS-maailma ovat erillään toisistaan ja näiden linkitys toisiinsa tapahtuu tekstimuotoisilla viitteillä joiden oikeellisuutta ei tarkisteta. (Styled-components 2018.)

Koodiesimerkissä 5 on toteutettu sama toiminnallisuus käyttäen apuna styled-components-kirjastoa. CSS-luokkien sekä erillisen, luokkia hallinnoivan react-komponentin sijaan luodaan suoraan react-komponentti, jolle asetetaan tyylit sekä tyylejä hallinnoiva logiikka luomisen yhteydessä parametrinä. Kaikki tapahtuu JS-tiedostossa merkkijonomuokkauksena, joten käytössä on kaikki JS-työkalut, datatyypit, funktiot ja ominaisuudet. Syntaksi on lyhyempää, keskitetympää ja selkeämpää. Tapa myös pakottaa erottamaan tyylikomponentin sekä sovelluslogiikan, joka on hyvä ratkaisu ylläpidettävyyden kannalta. (Styled-components 2018) Opinnäytetyön tekjiä havaitsi, että JS-varianttia, esimerkeiksi Typescriptiä, käytettäessä tyyli-deklaraatioon pystyy siirtymään, mikäli tekstieditori tukee muuttujien deklaraatioon siirtymistä.

```
import styled from ‘styled-components’

const MaybeRedMaybeCollapsedComponent =
  styled.span`
    width: 100px;
    ${props =>
      props.red ? ‘background-color: red;’ : ‘’}
    ${props =>
      props.collapsed ? ‘height: 0px;’ : ‘’}
`;
```

Koodiesimerkki 5: Dynaaminen CSS styled-components -kirjastoa käyttäen

6.6 React-virtualized-list

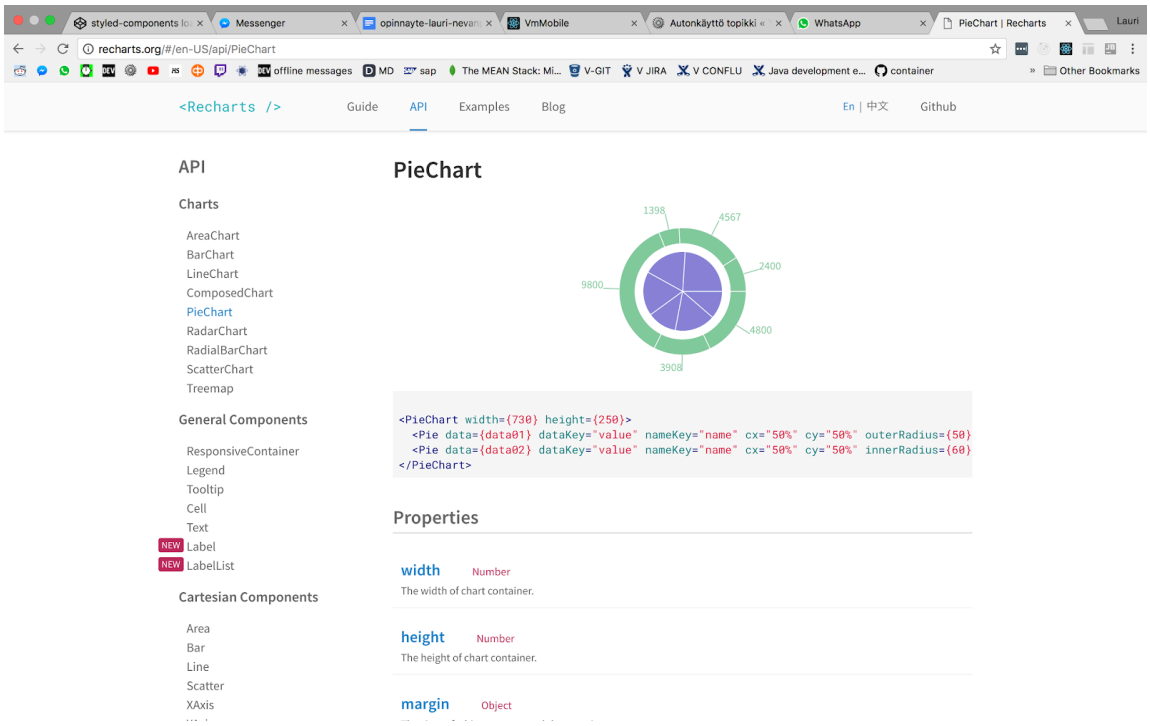
DOM-operaatiot, eli esimerkiksi uusien html-elementtien renderöinti, ovat selaimelle raskaita (Vukelic 2018). Yksi osa web-ohjelman optimointia on minimoida DOM-operaatiot. Tehdyssä tuntikirjausohjelmassa on lista, joka saattaa sisältää satoja elementtejä. Ilman optimointeja tälle sivulle siirryttäessä selain renderöisi siis pahimmassa tapauksessa satoja listaelementtejä kerralla, vaikka käyttäjän tarvitsisi nähdä kerralla vain kymmenen. Tämä johtaa pienitehoisilla laitteilla ohjelman jäätymiseen pahimmillaan muutamaksi sekunniksi, kun selain renderöi listan jokaisen elementin.

Käyttäjä näkee listasta kulloinkin vain noin 10 elementtiä, joten tehokasta olisi renderöidä kulloinkin vain ne elementit, jotka käyttäjä näkee. React-virtualized-list seuraa, missä kohdassa käyttäjä on listalla ja laittaa selaimen renderöimään vain ne elementit, jotka käyttäjä näkee. (React-virtualized-list-repositorio 2018.)

6.7 Recharts

Recharts on monipuolinen datan visualisointiin luotu avoimen lähdekoodin React-kirjasto. Koodiesimerkissä 7 esitetään syntaksi, jolla luodaan kaksitasoinen ympyrädiagrammi, joka on esitetty kuvassa 2.

Opinnäytetyön tekijä huomasi kirjastoa käyttäessään, että kirjaston ympyrädiagrammi on hyvin muokattavissa. Yksinkertaisten tyylittelyjen lisäksi diagrammia pystyi muokkaamaan matemaattisella piirtofunktiolla, joka mahdollisti työaikadiagrammin toteuttamisen sellaisena, kun se oltiin sovellukseen suunniteltu (Kuva 1).



KUVA 2. Recharts-ympyrädiagrammi

```
<PieChart
  width={730}
  height={250}
>
  <Pie
    data={data01}
    dataKey="value"
    nameKey="name"
    cx="50%"
    cy="50%"
    outerRadius={50}
    fill="#8884d8"
  />
  <Pie
    data={data02}
    dataKey="value"
    nameKey="name"
    cx="50%"
    cy="50%"
    innerRadius={60}
    outerRadius={80}
    fill="#82ca9d"
    label
  />
</PieChart>
```

KOODIESIMERKKI 7: Recharts-syntaksi ympyrädiagrammille

6.8 Käyttöliittymän muut teknologiavalinnat

Create-react-app

Käyttöliittymä luotiin komentorivityökalulla nimeltä create-react-app.
Perusasetuksilla työkalu luo yksinkertaisen react-projektin ja webpack-pohjaisen kehitysympäristön. Kehitysympäristöön kuuluu

- ES6-standardin JS:n muuttaminen standardiin jota selaimet ymmärtävät
- tehdyn ohjelmiston ja riippuvuuksien pakkaaminen yhteen JS-tiedostoon
- web-palvelin joka tarjoaa pakattua ohjelmistoa
- automaattinen selaimen päivitys, kun ohjelmistoon on tehty muutoksia ja selain on yhteydessä palvelimeen

Yarn-pakettihallinnoija

Yarn on Facebookin luoma pakettihallinnoija-komentorivityökalu.

Pakettihallinnoijan tehtävä on asentaa paketti (ohjelmakoodia jonka tarkoitus on tehdä jokin asia) julkisesta rekisteristä kehittäjän tietokoneelle. Jokainen paketti voi sisältää alipaketteja. Tavallinen projekti voi sisältää kymmeniä, satoja tai jopa tuhansia paketteja, alipaketteina sekä alipakettien alipaketteina. (McKenzie, Nagasawa & Kyle 2016.)

Yarn käyttää npm (Node packet manager) -pakettirekisteriä, ja se on luotu korvaajaksi npm-komentorivityökalulle. Facebook loi Yarnin, koska he eivät olleet tyytyväisiä npm-komentorivityökalun suorituskykyyn, turvallisuuteen eivätkä johdonmukaisuuteen. (McKenzie ym 2016.)

Suurimmat eroavaisuudet npm-komentorivityökaluun verrattuna ovat

- Välimuisti ladatuille paketeille. Jos työkalu on ladannut paketin kerran, pakettia ei enää tarvitse ladata.
- Algoritmi, joka varmistaa, että eri tietokoneilla tehdyt asennukset käyttävät varmasti samoja paketteja.

7 PALVELIMEN JS-KIRJASTOVALINNAT

Express

Express on Node JS -ympäristöön tehty palvelinviitekehys. Se on hyvin yksinkertainen kehys, jolle on tehty monia välikirjastoja, jotka helpottavat ohjelmointia.

Mongo DB

Mongo DB on NoSQL-tietokanta, joka tallentaa dataa BSON-muodossa. BSON on binäärimuotoinen esitys JSON-datasta. Tietokannan dataobjektit eivät ole tyypitettyjä. (MongoDb 2018.) Uudentyyppisiä objekteja voi luoda ilman tietokannan asetusten muuttamista, joten tietokanta on opinnäytetyön tekjän kokemuksen mukaan hyvä nopeaan ja kokeilevaan kehityskäyttöön. Tämä vapaus kuitenkin voi hankaloittaa kehitystyötä etenkin projektin edetessä, joten valitsin väliin Mongoose-kirjaston, joka abstrahoi MongoDB-operaatiot helpommin ylläpidettävään muotoon.

Mongoose luo tietokantaan tallennettaville dataobjekteille objektit, joiden kautta kaikki kyseistä objektia koskevat tietokantaoperaatiot kulkevat. Tämä varmistaa, että tietokantaan tallennettu data on konsistenttia. (Mongoose 2018.)

8 PALVELIMEN TEKNISET VALINNAT

8.1 Palvelimen vastausten määrittely

Palvelimen vastausten määrittäminen on tasapainoilua koon ja määrän kanssa. Liian isot vastaukset tekevät sivun toiminnasta hidasta etenkin laitteella jossa on huono verkkoyhteys, sillä laite joutuu hakemaan suuren määrän turhaa dataa kutsujen yhteydessä. Esimerkkinä palvelin, joka palauttaisi samassa vastauksessa käyttäjän tiedot sekä listan käyttäjän tekemistä tuntimerkinnöistä. Aina kun käyttäjän tietoihin tulisi muutos palvelimella, käyttöliittymäsovellus joutuisi hakemaan päivitettyjen tietojen lisäksi myös käyttäjän tekemät tuntimerkinnät.

Jos taas vastaukset ovat liian pilkottuja, esimerkiksi niin, että jokainen käyttäjän tietojen kenttä on oman päätepisteensä takana, palvelimelle voi joutua tekemään käyttöliittymästä kymmeniä, ellei satoja kutsuja ennen kuin käyttöliittymä voi renderöidä näkymän.

Tuntikirjaussovelluksessa pyrittiin minimoimaan tehtyjen kutsujen määrä, ja toisaalta pyrittiin myös pitämään palvelinvastaukset pieninä.

- Kaikki tuntimerkinnät palautetaan aina yhtenä vastauksena ohjelman käynnistyksen yhteydessä, jotta mahdolliset muutokset näkyisivät käyttöliittymässä. Jotta vastauksesta ei tule liian iso, säilytetään palvelimella vain palkanlaskentaohjelmistolle lähettämättömät merkinnät sekä lähetetyt merkinnät kahden viikon ajalta.
- Lisättäessä tuntimerkintää, palvelin palauttaa luodun tuntimerkinnän ja mahdolliset lisäyksen myötä muuttuneet tuntimerkinnät. Muut tuntimerkinnät voivat muuttua, mikäli tuntimerkintä on limittäin toisen merkinnän kanssa, jolloin molemmat merkinnät merkataan konfliktoituneiksi merkinnöiksi.
- Käyttäjälle merkatut projektit saadaan yhdellä kutsulla ohjelman käynnistyksen yhteydessä. Projekteihin viitataan uniikilla tunnisteella tuntimerkinnöissä.

Käyttöliittymän täytyy siis käynnistyksen yhteydessä hakea kaikki tunnit sekä projektit, jotta tieto on varmasti ajan tasalla. Tämän jälkeen palvelin palauttaa vain yksittäisiä tuntimerkintöjä, jotka päivitetään ohjelman tilaan uniikin tunnisteen perusteella. Käyttäjä voi halutessaan hakea palvelimelta kaikki tuntimerkintänsä, mikäli hän tietää tehneensä muutoksia toisella laitteella.

8.2 JSON Web Token

JSON Web Token on standardi turvalliseen tietojen siirtoon kahden osapuolen välillä JSON-objektin muodossa. Se on määritelty standardissa RFC-7519. JSON-data on salattu joko julkinen-yksityinen -avainparin avulla tai HMAC-tekniikalla, jossa JSON Web Token salataan ja puretaan palvelimessa olevan avaimen avulla. (Jones, Bradley & Sakimura 2015.) Tuntikirjausjärjestelmä käyttää jälkimmäistä tapaa.

Kun käyttäjä kirjautuu onnistuneesti sisään, palvelin lähettää vastauksessa käyttäjälle JSON Web Tokenin, joka sisältää tunnisteen, jolla käyttäjän tiedot on tallennettu tietokantoihin. Jokaisessa tulevassa palvelinkutsussa täytyy asettaa annettu JSON Web Token kutsun otsikkotietueen Authorization-kenttään Bearer-etuliitteellä, kuten RFC-6750 -standardissa on märitelty (Jones & Hardt 2012).

9 OHJELMOINTITYYLI JA ARKKITEHTUURI

9.1 Yhteiset ratkaisut

Ohjelmointikieli

Opinnäytetyön käyttöliittymä sekä palvelinohjelmisto on ohjelmoitu ES6 (ECMAScript 6) -standardin JS:llä.

ES on standardi, jonka JS toteuttaa (Aranda 2017). ES-standardi on kehittynyt vuosien aikana, ja selainvalmistajat toteuttavat selaintukea omaan tahtiinsa. ES6 on toiseksi uusin versio standardista, eikä se ole yleisesti tuettu selaimissa. Kirjoitettu ohjelmakoodi käännetään vanhempaan ES-standardiin webpack-kääntäjällä. ES6 tuo monia parannuksia verrattuna vanhempiin ES-versioihin. Näistä suurimmat ovat

- lambda-funktiot
- sapluunamerkkijonot (template string)
- vakiomuuttujat
- levitysoperaattori (spread operator)

Ominaisuuksista oli paljon apua ohjelmoitaessa. Opinnäytetyön tekijän kokemuksen mukaan ominaisuudet helpottivat monia asioita: Käyttöliittymän tyylikirjasto styled-componentsia ei voisi käyttää ilman sapluunamerkkijonoja. Tilanhallintakirjasto Reduxin käyttö oli vaivattomampaa levitysoperaattoria käyttäen. Lambda-funktiot tekevät ohjelmakoodista paljon selkeämpää, kun anonyymi funktio voidaan luoda huomattavasti lyhyemmällä syntaksilla.

Funktionaalinen ohjelmointi

Opinnäytetyön kirjoittaja on aikaisemmin tutustunut funktionaaliseen ohjelmointiin Elm-ohjelmointikielen kanssa. Elm on vahvasti tyypitetty funktionaalinen käyttöliittymäohjelmointikieli, joka kääntyy JS:ksi. (Elm-lang 2018.)

Funktionaalinen ohjelmointi on ohjelmointiparadigma, jossa suositaan puhtaita funktioita, funktioiden rakentamista muista funktioista sekä deklaratiivista ohjelmointimallia. Paradigmassa vältetään jaettua, sekä muuttuvaa tilaa. (Elliot 2017.)

Etenkin käyttöliittymän kirjastovalinnat soveltuivat hyvin funktionaaliseen ohjelmointiin. Opinnäytetyön tekijälle tuli yllätyksenä, että React + Redux -arkkitehtuuri vastasi lähes täysin Elm-ohjelmointikielen käyttöliittymä- sekä tilanhallinta-arkkitehtuuria.

9.2 Käyttöliittymä

Mobiili ensin

Mobiili ensin (mobile first) -lähestymistapa käyttöliittymäsuunnitteluun tarkoittaa sitä, että sovellus suunnitellaan ensiksi pienimmälle tuetulle näyttökoolle, ja tämän pohjalta suunnitellaan suuremmat näyttökoot. Kokemusten perusteella käyttöliittymä on helpompi skaalata pienestä näytöstä suurempaan, kuin suuresta näytöstä pienempään. (Mayven n.d.)

Tiukan aikataulun vuoksi työaikasovellukseen toteutettiin vain mobiilikäyttöliittymä. Suunnitteilla oli oma käyttöliittymänsä myös työpöytälaitteille, mutta aikataulun vuoksi se jätettiin toteuttamatta. Tulevaisuudessa työpöytäkäyttöliitymän voi toteuttaa esimerkiksi yhdistämällä työaikalistasivun sekä työajanseurantasivun yhteen sivuun.

Yhteydettömyys ensin

Yhteydettömyys ensin -ajattelussa verkkoyhteys ajatellaan enemmän sovelluksen lisäksi, kuin suunnittelun lähtökohdaksi. Suunnittelu pohjaa paikallisen tallennustilan hyödyntämiseen palvelinkutsujen sijaan kun verkkoyhteyttä ei ole saatavilla. (Teixeira 2016.)

Käyttöliittymä suunniteltiin käyttäen lähtökohtana yhteydettömyys ensin (offline first) -ajattelua. Yhteydettömyys ensin -ajattelu sopii mobiilissa käytetävään tuntiseurantaohjelmaan hyvin, sillä tuntimerkinnät vaikuttavat laskutukseen ja ovat täten tärkeitä. Käyttäjä ei monesti muista muutaman päivän takaisia merkintöjä, joten on hyvä, että ne tallentuvat sovellukseen vaikka internet-yhteyttä ei ole saatavilla.

Tuntikirjaussovelluksessa yhteydettömyys ensin toteutettiin suunnittelemalla palvelimen päätepisteitten sekä käyttöliittymän tietorakenne mahdollistamaan yhteydetön toiminta.

Käyttöliittymässä käytettiin redux-persist kirjastoa, joka tallentaa ohjelman tilan paikallisesti, jolloin edellisen istunnon tila on saatavilla, kun ohjelman käynnistää seuraavan kerran. (Redux-persist -versionhallintarepositorio 2018.)

Arkkitehtuuri

React-ohjelmakoodi on deklaratiivistä, eli ohjelmakoodissa kerrotaan, minkälainen käyttöliittymä rakennetaan ohjelman kulloisenkin tilan perusteella. Imperatiivisessa ohjelmakoodissa taas ohjelmakoodi kertoo, mitä muutetaan, kun jotain tapahtuu. Deklaratiivista ohjelmointiä käytetään usein funktionaalisessa ohjelmoinnissa, kun taas imperatiivinen on yleisempää objektiorientoituneessa ohjelmoinnissa (Mundy 2017). Koodiesimerkissä 8 tehdään sama asia deklaratiivisesti ja imperatiivisesti:

Imperatiivinen:

```
jos tapahtuma on ‘avaa info-popup’:
  käske popup-hallinnoijaa avaamaan popup-ikkuna
  aseta popup-tyypiksi ‘info’
  aseta popup-tekstiksi annettu teksti
```

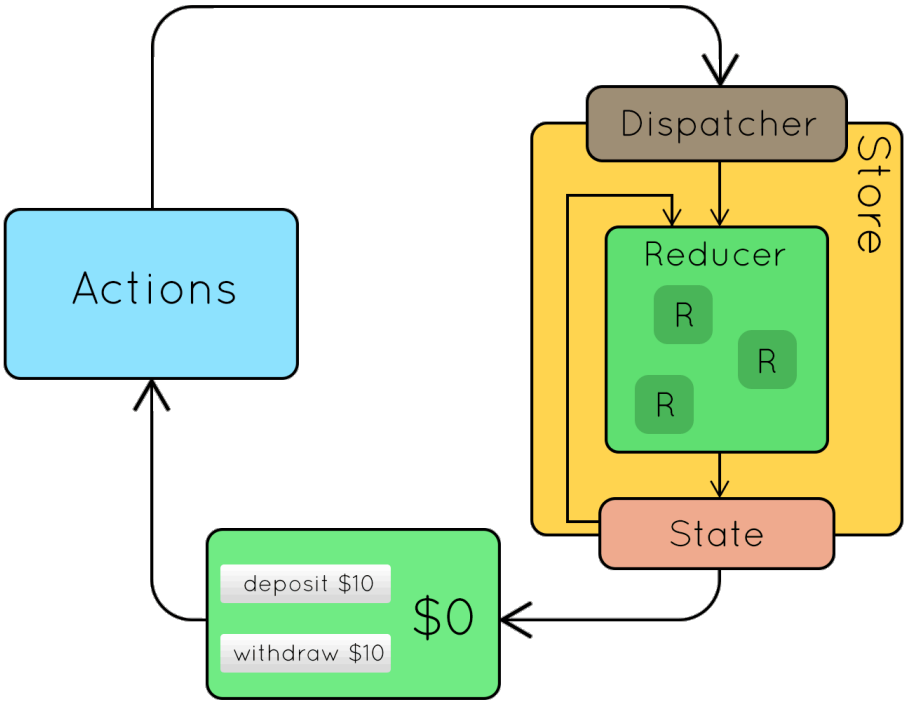
Deklaratiivinen:

```
jos tapahtuma on ‘avaa info-popup’:
  aseta ohjelman popup-tilaksi {
    popupNäkyvissä: totta,
    popupTyyppi: ‘info’,
    popupTeksti: annettu teksti
  }
```

KOODIESIMERKKI 8: Imperatiivisen- ja deklaratiivisen ohjelmoinnin eroavaisuudet

Imperatiivisessa ohjelmoinnissa siis ohjelmoija itse askel askeleelta rakentaa popup-elementin. Deklaratiivisessa esimerkissä asetetaan vain tila, jonka ohjelma muuntaa näkymäksi.

Opinnäytetyön tekijän kokemuksen mukaan deklaratiivisuus tuo muutamia etuja, joista suurimmat edut ovat tilanhallinnassa. Mikäli ohjelman käyttöliittymä on funktio, joka muuttaa tilan näkymäksi kuten Reactissa, niin tätä tilaa on erittäin helppo seurata sekä muokata. Mahdolliset virhetilanteet ovat helpompia selvittää, sillä yksi tila vastaa yhtä käyttöliittymän näkymää ja tilan perusteella voi helposti päätellä ongelmakohtia.



KUVIO 3. React-Redux -arkkitehtuuri (Pytel & Terpil 2016)

Kuvio 3 kuvaa React-Rexux -sovelluksen tilan päivittymistä.

1. React-funktio muuntaa redux-tilan react-elementeiksi, jotka virtual dom luo käyttäjän selaimeen. Dom-elementteihin on asetettu attribuuteiksi funktioita, jotka laukaisevat eri tyyppisiä redux-tapahtumia (Actions).
2. Reducer-funktio ottaa ohjelman tilan sekä laukaistun tapahtuman ja palauttaa uuden tilan, joka on luotu vanhan tilan sekä laukaistun tapahtuman pohjalta.
3. Reducer-funktion palauttama uusi tila asettuu reduxin tilaksi.
4. Sykli alkaa alusta.

10 KOULUTUS

10.1 Tausta

Tuntikirjaus-sovelluksen tekemisen aikana opitut tiedot sekä käytännöt tuli siirtää yrityksen muille työntekijöille. Tiedonvälityksen teemana oli käytännönläheisyys ja innostavuus. Tiedonvälityksessä ei käytetty kannustimia vaan opetuksesta sekä oppimismateriaaleista pyrittiin tekemään sellaisia, että ne motivoisivat työntekijöitä aloittamaan omia harrasteprojekteja. Esimerkiksi versionhallintaan tehtiin valmis projektipohja, josta työntekijät voisivat helposti saada projektipohjan, joka sisältää opetetut teknologiat.

10.2 Koulutuksen järjestäminen

Toimeksiantajan toimistolla alettiin järjestämään koodiaamiainen-tapahtumaa. Tapahtumassa on tarjolla aamiainen ja noin puolen tunnin käytännönläheinen puhe. Koodiaamiainen saatettiin pohjustaa viidentoista minuutin esityksellä muutamaa päivää ennen tapahtumaa. Tässä esityksessä käytiin läpi tulevaan koodiaamiaiseen liittyvää asiaa.

Ensimmäisissä kahdessa koulutuksessa on tarkoitus antaa tarvittavat perustiedot modernista javascriptistä, perusteet Reactista ja tieto kirjastoista, joita Reactin kanssa kannattaa käyttää. Tämän jälkeen luennoissa ohjelmoidaan reaaliaikaisesti React-sovellusta, ja tarkoitus on tuoda ilmi, miten helposti näkyvää tulosta syntyy.

10.3 Koulutuskerrat

Ensimmäinen koulutus

Ensimmäinen koulutus alustettiin käymällä läpi JS-ekosysteemin nykytila. Alustuksessa tarkasteltiin eri kieliä, jotka kääntyvät JS:ksi, sekä tutkittiin hieman Typescriptin sekä React-kirjaston perusteita.

Itse koulutuksessa käytiin läpi React-kirjaston toimintaa syvemmin, ja tutustuttiin Reactin kanssa hyväksi havaittuihin kirjastoihin ja niiden perusteisiin. React-kirjastosta koulutuksessa käytiin läpi React-komponentit, -elementit sekä Virtual DOM. Apukirjastoista käytiin läpi Redux sekä Reselect.

Toinen koulutus

Toinen koulutus pyrittiin pitämään mahdollisimman lyhyenä. Siinä käytiin style-components -kirjaston perusteet, luotiin React-projekti Typescript-kielellä create-react-app -komentorivityökalulla ja korvattiin luodun projektin CSS-tiedostot styled-components -komponenteilla.

Kolmas koulutus

Kolmannessa koulutuksessa ohjelmoitiin yksinkertainen ‘deittisovellus’, joka haki mallidatana kuvia sekä generoituja puhelinnumeroita www.randomuser.me -apista. Tämän koulutuksen yhteydessä lanseerattiin Github-repositiorio, jossa deittisovellus oli toteutettu käyttöliittymää lukuunottamatta. (https://github.com/Wombbu/react-redux-styled-components-template) Tällöin myös lanseerattiin kaikkien koulutusten powerpoint-esitykset.

11 POHDINTA

Kaiken kaikkiaan opinnäytetyön tekeminen sujui hyvin. Teknisellä saralla muutama pieni asia olisi kaivannut parantamista. Opinnäytetyö tehtiin JS:n ES6-versiolla, mutta jälkikäteen tuntuu siltä, että TypeScript olisi ollut parempi vaihtoehto, sillä staattisesti tyypitetty ohjelmistoprojekti on helpompi ylläpitää. Muuten tekniset valinnat osoittautuivat hyviksi, ja ohjelmiston teon jälkeen olen päässyt tekemään asiakkaalle projektia React, Redux sekä styled-components -kirjastoilla, eli juuri niillä teknologioilla, joita opinnäytetyössä käytettiin.

Opetus meni mielestäni hyvin. Esitykset sujuivat sulavasti, ja jokaisen opetuskerran jälkeen paikalle jäi useita ihmisiä keskustelemaan aiheesta. Koulutuksista tullut palaute oli positiivista, ja sain kolme yksityisviestiä, jossa kyseltiin opetusmateriaaleja. Tältä osin olen tyytyväinen lopputulokseen.

Itse tuntikirjausjärjestelmästä tuli hyvä, ja ennen kaikkea pystyin käyttämään siinä erittäin laajalla skaalalla itselleni uusia teknologioita. Tuntikirjausohjelmaa ei vielä tämän raportin kirjoitushetkellä ole otettu käyttöön. Tämä on harmi, mutta se ei haittaa itseäni niin paljoa, sillä projektin ensisijainen tarkoitus olikin oppia uutta ja välittää tästä tietoa. Nämä tavoitteet täyttyivät mielestäni erittäin hyvin.

LÄHTEET

Abramov, D. 2015. React Components, Elements and Instances. Luettu 10.12.2017
https://reactjs.org/blog/2015/12/18/react-components-elements-and-instances.html

Aiyer, A. 2016. Redux Selector Pattern. Luettu 27.12.2017
https://gist.github.com/abhiaiyer91/aaf6e325cf7fc5fd5ebc70192a1fa170

Angular.js-versionhallintarepositorio. N.d. CHANGELOG. Luettu 16.11.2017.
https://github.com/angular/angular.js/blob/master/CHANGELOG.md

Brown, P. 2017. State of the Union: npm. Luettu 14.11.2017.
https://www.linux.com/news/event/Nodejs/2016/state-union-npm

CoffeeScript-versionhallintarepositorio. N.d. List of languages that compile to JS. Luettu 14.11.2017.
https://github.com/jashkenas/coffeescript/wiki/list-of-languages-that-compile-to-js

Elliot, E. 2017. Master the JavaScript Interview: What is Functional Programming?. Luettu 14.2.2018
https://medium.com/javascript-scene/master-the-javascript-interview-what-is-functional-programming-7f218c68b3a0

Elm-lang. 2018. Guide. Luettu 14.2.2018
https://guide.elm-lang.org/

Havery, C. 2014. Javascript: the weird parts. Luettu 12.11.2017.
https://charlieharvey.org.uk/page/javascript_the_weird_parts

Jones, M., Bradley, J. & Sakimura, N. 2015. JSON Web Token (JWT). Luettu 6.3.2018.
https://tools.ietf.org/html/rfc7519

Jones, M. & Hardt, D. 2012. The OAuth 2.0 Authorization Framework: Bearer Token Usage. Luettu 6.3.2018.
https://tools.ietf.org/html/rfc6750

Jones, E. 2013. Static type checking: More productive for large projects
http://www.evanjones.ca/static_type_checking.html

Mayven N.d. Mobile First Design: Why It's Great and Why It Sucks. Luettu 22.2.2018
https://mayvendev.com/blog/mobilefirst

Medium 2016a. Managing your React state with Redux. Luettu 20.12.2017
https://medium.com/the-web-tub/managing-your-react-state-with-redux-affab72de4b1

Medium 2016b. Time Travel in React Redux apps using the Redux DevTools. Luettu 20.12.2017
https://medium.com/the-web-tub/time-travel-in-react-redux-apps-using-the-redux-dev-tools-5e94eba5e7c0

Aranda, M. 2017. What's the difference between JavaScript and ECMAScript? Luettu 14.2.2018
https://medium.freecodecamp.org/whats-the-difference-between-javascript-and-ecmascript-cba48c73a2b5

MongoDb. 2018. JSON and BSON. Luettu 1.2.2018.
https://www.mongodb.com/json-and-bson

Mongoose. 2018. Guide. Luettu 19.1.2018.
http://mongoosejs.com/docs/guide.html

Mundy, I. 2017. Declarative vs Imperative Programming. Luettu 4.3.2018.
https://codeburst.io/declarative-vs-imperative-programming-a8a7c93d9ad2

Newton, M. 2017. The Definitive Guide to Redux Persist. Luettu 20.12.2017
https://blog.reactnativecoach.com/the-definitive-guide-to-redux-persist-84738167975

Peyrott, S. 2016. A Brief History Of JavaScript. Luettu 12.11.2017.
https://auth0.com/blog/a-brief-history-of-javascript/

Pytel, V., Terpil, I. 2016. Redux. From Twitter hype to production. Luettu 4.3.2018.
http://slides.com/jenyaterpil/redux-from-twitter-hype-to-production#/

React-versionhallintarepositorio. N.d. CHANGELOG. Luettu 16.11.2017.
https://github.com/facebook/react/blob/master/CHANGELOG.md

React-virtualized-list -versionhallintarepositorio. N.d. 2018. README. Luettu 4.1.2018
https://github.com/developerdizzle/react-virtual-list

Redux-persist -versionhallintarepositorio. N.d. 2018. README. Luettu 1.3.2018.
https://github.com/rt2zz/redux-persist

Redux-versionhallintarepositorio. N.d. Prior Art. Luettu 20.12.2017
https://github.com/reactjs/redux/blob/master/docs/introduction/PriorArt.md

Reselect-versionhallintarepositorio. 2018. README. Luettu 22.1.2018.
https://github.com/reduxjs/reselect#motivation-for-memoized-selectors

McKenzie, S., Nakazawa, C. & Kyle, J. 2016. Yarn: A new package manager for JavaScript. Luettu 4.1.2018
https://code.facebook.com/posts/1840075619545360

Stack Overflow Developer Survey. 2017. Luettu 14.11.2017.
https://insights.stackoverflow.com/survey/2017#technologies-and-occupations

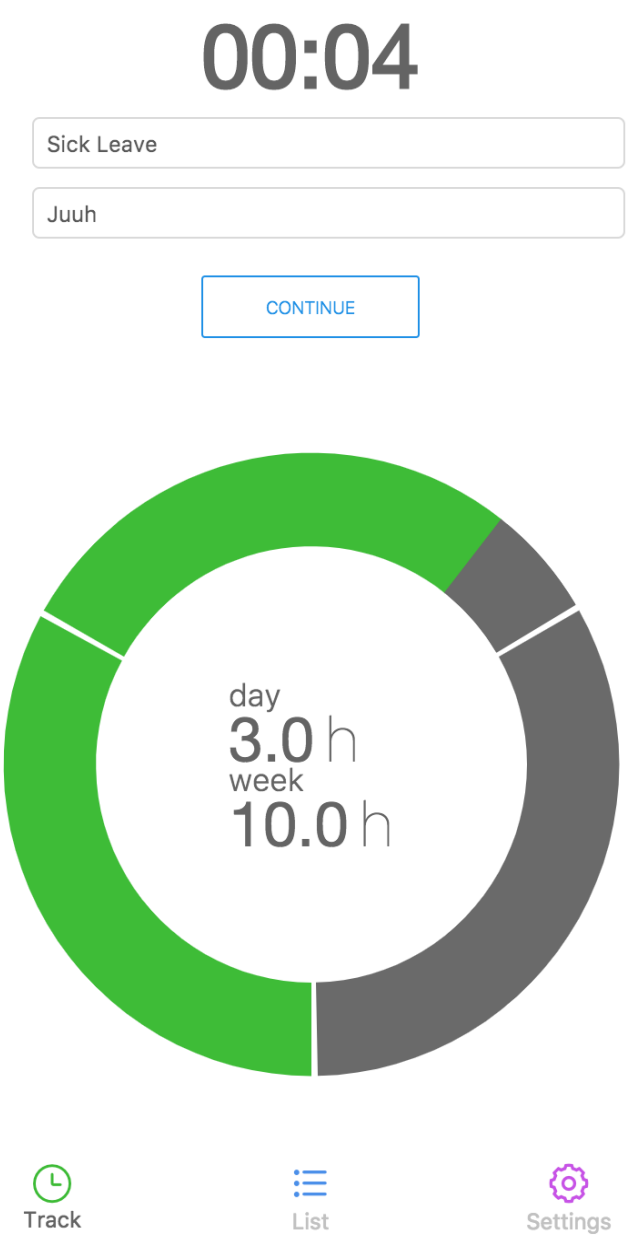
Styled-components. 2018. Basics. Luettu 1.2.2018.
https://www.styled-components.com/docs/basics

Teixeira, P. 2016. Build More Reliable Web Apps with Offline-First Principles. Luettu 22.2.2018
https://thenewstack.io/build-better-customer-experience-applications-using-offline-first-principles/

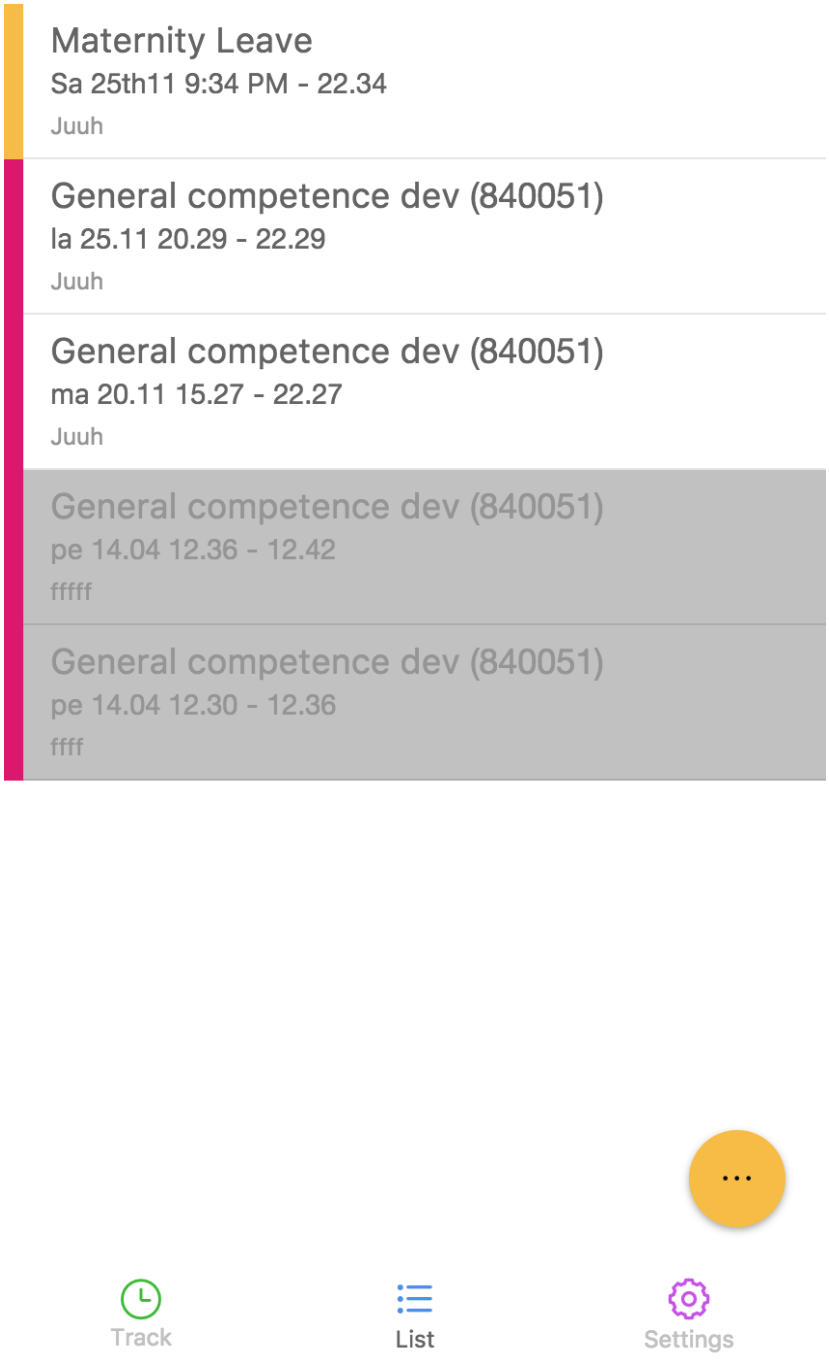
Vukelic, H. 2018. Taming huge collections of DOM nodes. Luettu 4.1.2017
https://codeburst.io/taming-huge-collections-of-dom-nodes-bebafdba332

LIITTEET

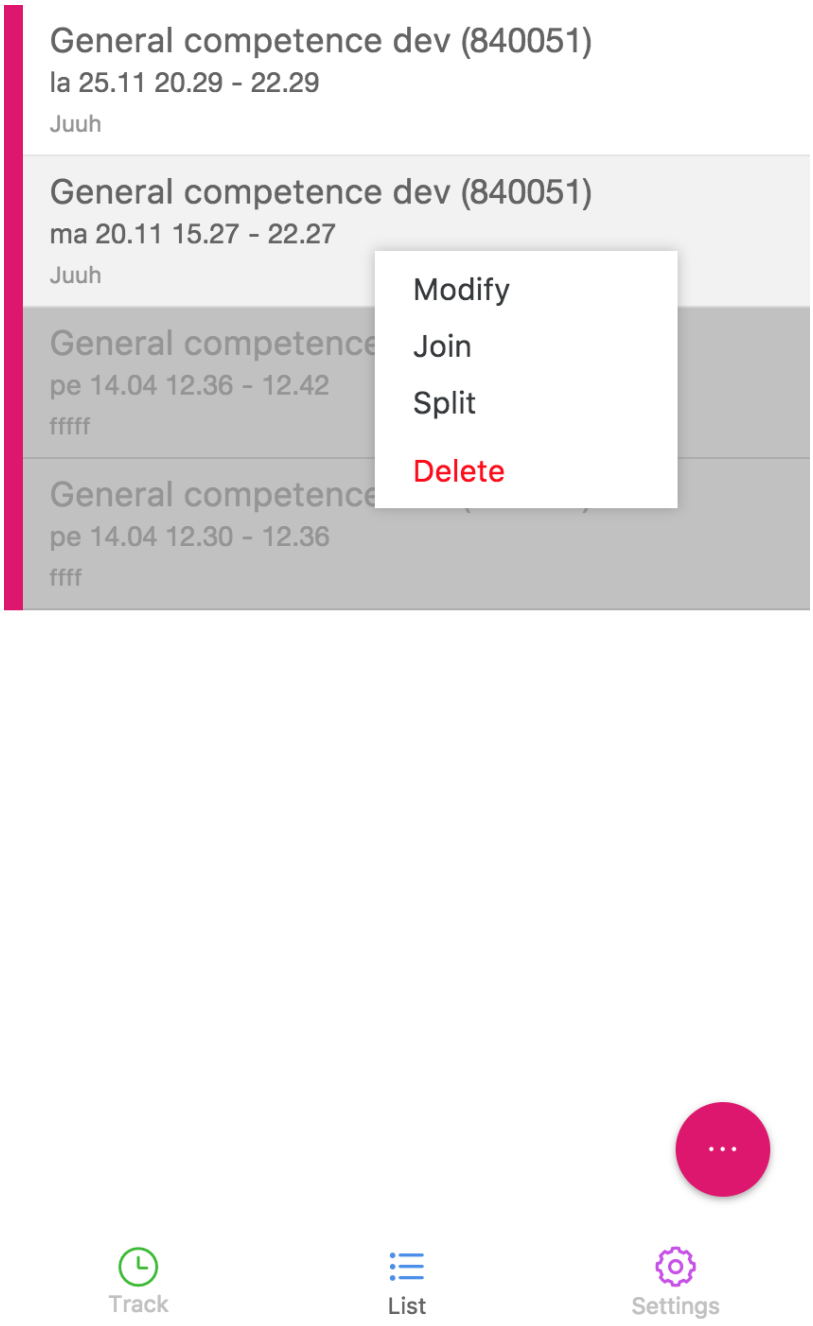
Liite 1. Luodun tuntkirjausjärjestelmän oletusnäkymä



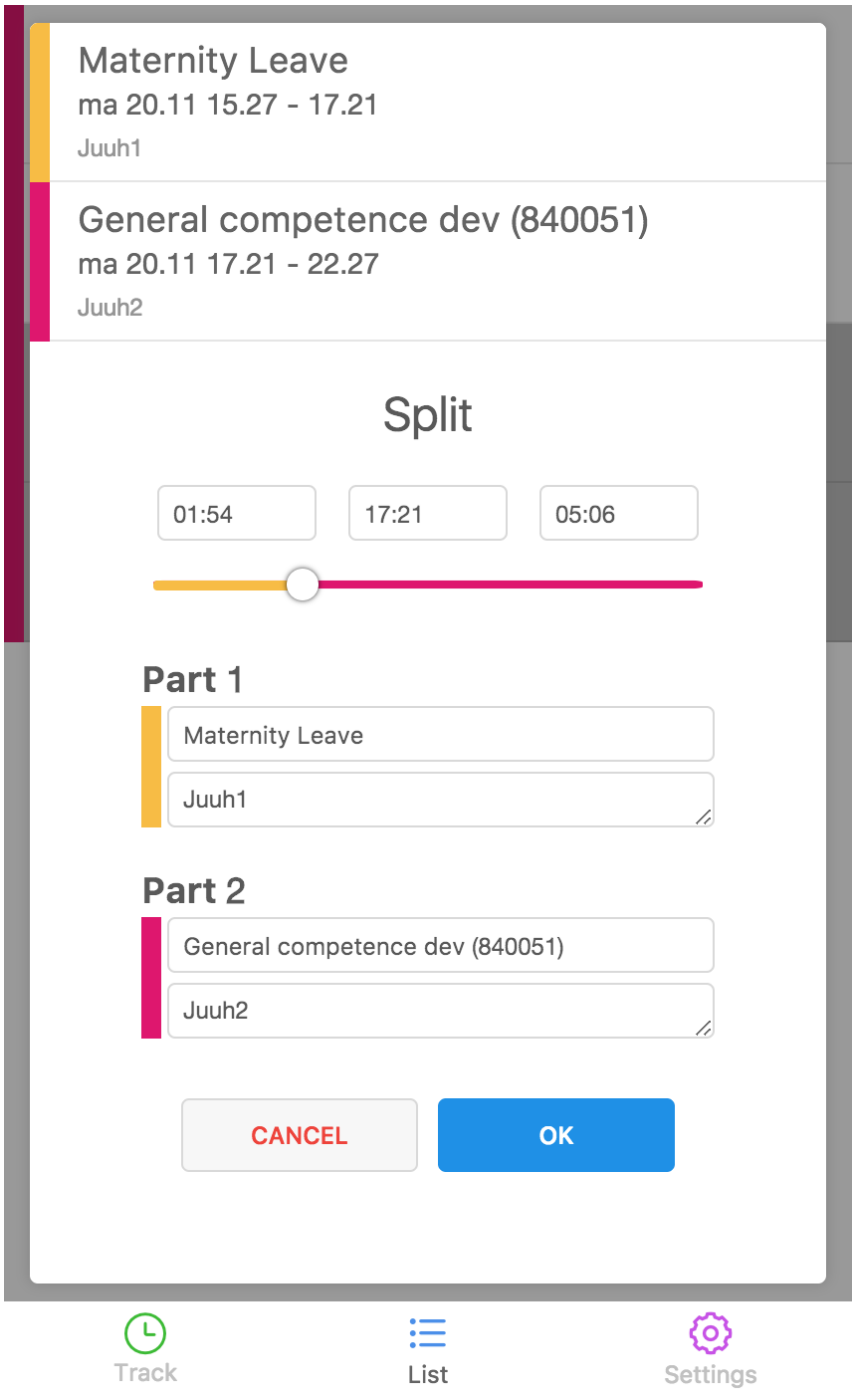
Liite 2. Luodun tuntikirjausjärjestelmän listanäkymä



Liite 3. Luodun tuntikirjausjärjestelmän listanäkymän muokkausoperaatiot



Liite 4. Luodun tuntikirjausjärjestelmän tuntien jakonäkymä



Liite 5. Luodun tuntikirjausjärjestelmän asetusnäkymä